**Compiler Design (BCST-602)**

**Important Questions**

**For First Sessional**

1. **Differentiate** between compilers and interpreters.
2. **Describe** language denoted by the following regular expression: (1 + 0)\*
3. **Express** the main idea of NFA? And discuss with examples (a/b)\*
4. **What** is the role of lexical analyzer? **Enumerate** the issues handled by a lexical analyzer.
5. **What** is cross-compiler? **How** is boot-strapping of a compiler done to a second machine?
6. Give the parse tree for the statement a: b \* c + 60.
7. Give the parse tree for the statement a:= b + c – (d+e).
8. Differentiate between Recursive Decent Parsing and Predictive Parsing.
9. Consider the following grammar and construct CLL(1) parsing table:

**S** → **i E t SS’ | a**

**S’** → **eS | ε**

**E** → **b**

1. **Show** the construction of NFA for following regular expression:

(a | b)\*a(a | b)(a | b)

1. **Define** Lex and Lex specifications. **How** lexical analyzer is constructed using lex? Give an example.
2. **Eliminate** all left recursions from the grammar given below”

S →Af | b

A→Ac | Sd | Be

B→ Ag | Sh | k

1. **Compute** FIRST (α) and FOLLOW(A) for the following given grammar:

E → TE’

E’ → +TE’ | ε

T → FT’

T’ → \*FT’ | ε

F → ( E ) | id

1. **Describe** the Input buffering techniques in detail.
2. **Point out why** is buffering used in lexical analysis? **What** are the commonly used buffering methods?
3. **Convert** the regular expression (a/b)\* abb (a/b)\* into NFA using Thompson construction.
4. **Explain how** LEX tool may be used to create lexical analyser
5. **How** is Finite Automata useful for lexical analysis? **Construct** the NFA and DFA for the following regular expression: ( a + b )\*abb
6. **Prove** that whether the following grammar is LL(1) or not?

S 🡪 iEtSS’ | a

S’🡪 eS | ε

E 🡪 b

1. **What** are shift/reduce and reduce/ reduce conflicts of LR(0) parser? **Explain** with examples.
2. Find the First() and Follow(A) of the each of the Non-terminal in the grammar given below:

**S→ aABe**

**A→Abc | b**

**B→ d**

1. Given the grammar G below, do the rightmost derivation in reverse and draw the parse tree for the Input String: id+id

**E → E + T | T**

**T → T \* F | F**

**F → (E) | id**

1. Differentiate between Shift/Reduce and Reduce/Reduce conflicts with examples.
2. Differentiate between Syntax Tree and Annotated Syntax Tree with examples.
3. Differentiate between Parse Tree and Syntax Tree with examples.
4. Consider the grammar given below:

S →ABCDE

A → a|ε

B → b|ε

C → c

D → d|ε

E → e|ε

1. **Calculate** FIRST () AND FOLLOW() sets.
2. **Eliminate** the left-recursion and do left-factoring of the following grammar:

S → ( S ) |S S | ( **)**

1. **What** is “Dangling-Else” problem? **How** do you remove it? Explain with examples.
2. **Explain** handle and handle pruning with examples.